Bolt Beranek and Newman Inc. (12)

JL

LEVEL II

Report No. 4182

ADA086123

Development of a Voice Funnel System

Quarterly Technical Report No. 4
1 May 1979 to 31 July 1979

June 1980

DTIC
ELECTE
S JUL 1 1980 D
A

Prepared for:
Defense Advanced Research Projects Agency

DDC FILE COPY

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

80 6 30 073

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER / 2. GOVT ACCESSION NO. AD-A086 123	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) (6) DEVELOPMENT OF A VOICE FUNNEL SYSTEM. ~~QUARTERLY TECHNICAL REPORT NO. 4~~	5. TYPE OF REPORT & PERIOD COVERED Quarterly Technical
	6. PERFORMING ORG. REPORT NUMBER 4182
7. AUTHOR(s) (10) M./Hoffman	8. CONTRACT OR GRANT NUMBER(s) (15) MDA903-78-C-0356, ARPA Order-3653
9. PERFORMING ORGANIZATION NAME AND ADDRESS Bolt Beranek and Newman Inc. 50 Moulton Street, Cambridge, MA 02138	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS ARPA Order No. 3653
11. CONTROLLING OFFICE NAME AND ADDRESS Defense Advanced Research Projects Agency 1400 Wilson Blvd., Arlington, VA 22209	12. REPORT DATE June 1980
	13. NUMBER OF PAGES 28
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) (11) Jun 80 (12) 31	15. SECURITY CLASS. (of this report) Unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Distribution Unlimited	
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report) (9) Quarterly rept. no. 4, 1 May-31 Jul 79,	
18. SUPPLEMENTARY NOTES (14) BBN-4182	
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Voice Funnel, Digitized Speech, Packet Switching, Butterfly Switch, Multiprocessor	
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) This Quarterly Technical Report covers work performed during the period noted on the development of a high-speed interface, called a Voice Funnel, between digitized speech streams and a packet-switching communications network.	

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

060100 slt

Report No. 4182 Bolt Beranek and Newman Inc.

DEVELOPMENT OF A VOICE FUNNEL SYSTEM

QUARTERLY TECHNICAL REPORT NO. 4
1 May 1979 to 31 July 1979

June 1980

This research was sponsored by the
Defense Advanced Research Projects
Agency under ARPA Order No.: 3653
Contract No.: MDA903-78-C-0356
Monitored by DARPA/IPTO
Effective date of contract: 1 September 1978
Contract expiration date: 30 November 1980
Principal investigator: R. D. [illegible]ettberg

Accession For
NTIS GRA&I
DDC TAB
Unannounced
Justification
By
Distribution/
Availability Codes
Dist Avail and/or special
A

Prepared for:

Dr. Robert E. Kahn, Director
Defense Advanced Research Projects Agency
Information Processing Techniques Office
1400 Wilson Boulevard
Arlington, VA 22209

The views and conclusions contained in this document are those of the author and should not be interpreted as necessarily representing the official policies, either express or implied, of the Defense Advanced Research Projects Agency or the United States Government.

QUARTERLY TECHNICAL REPORT 4

Contents

1. Introduction

This Quarterly Technical Report, Number 4, describes aspects of our work performed under Contract No. MDA903-78-C-0356 during the period from 1 May 1979 to 31 July 1979. This is the fourth in a series of Quarterly Technical Reports on the design of a packet speech concentrator, the Voice Funnel.

During this quarter we have completed the basic design of the Processor Node hardware for the Butterfly Multiprocessor. This report concentrates on the Memory Management section of that design.

A major change has occurred in the design of the Processor Node since this report was first drafted. As a result of delivery problems with the Z8000 and the anticipated availability of the Motorola MC68000, we have changed the design of the Processor Node to use the Motorola microprocessor. The technical information in this report reflects this change.

We anticipate that the change from the Zilog Z8000 to the Motorola MC68000 will have some important advantages. The MC68000 is more homogeneous in its address space and addressing modes. The bus organization of the MC68000 should permit recovery after a memory accessing failure, although the initial design of the MC68000 will not always provide recovery. Perhaps one of the most important characteristics of the MC68000 is its acceptance by much of the research community. We can expect that this will ease the design of many tools and assure a long life for the Butterfly Multiprocessor.

2. Memory Architecture

The computing power of a machine depends as much on the architecture of its memory system as it does on the architecture of its processor.

Three considerations dominate the design of the memory system of the Butterfly Multiprocessor: 1) the software of this machine is based on "processes" (rather than the strips of the Pluribus); 2) it will support large and complex software, requiring protection and separation between the component pieces; and 3) it is a tightly-coupled multiprocessor, so that shared memory will be an important form of communications.

The virtual memory system of the Butterfly Multiprocessor provides each process with up to 256 memory segments. Each segment can be from 256 to 64K bytes long. Each segment has individual protection and relocation attributes, and each may represent local memory (on the same Processor Node), remote memory, or I/O device registers.

2.1 Address Spaces

The Butterfly Multiprocessor has a 32-bit physical address space. This address space is the concatenation of the physical address spaces of all the Processor Nodes in the machine. The physical address space of each Processor Node consists in turn of 8 subspaces. Each subspace is 1M bytes long. The 32-bit physical address is organized as an 8-bit Processor Node number (permitting up to 256 Processor Nodes), a 3-bit subspace number, an unused bit, and a 20-bit subspace offset. This address format is illustrated in Figure 2.1-1.

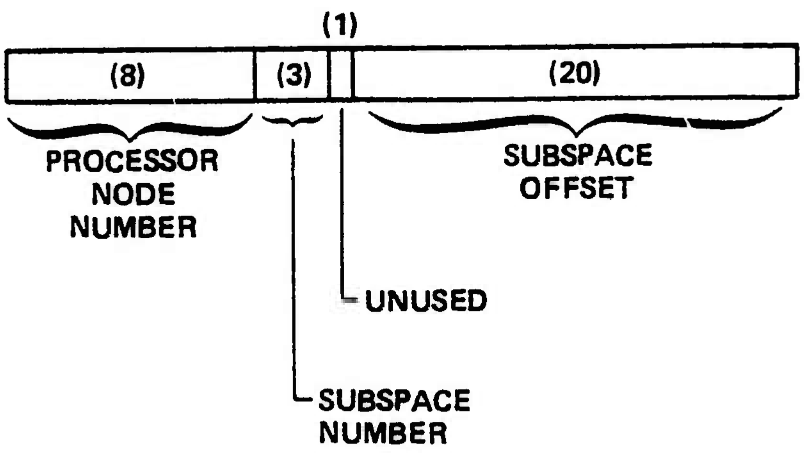

Figure 2.1-1 Physical Address Format

The 8 physical address subspaces of a Processor Node are defined as follows:

Subspace Number	Subspace Defined
0	Local EPROM memory
1	I/O Device Registers
2	Bulk memory not via switch
3	Bulk memory via switch
4	Memory Management SAR
5	Control Register Set A
6	Control Register Set B
7	Control Register Set C

- Subspace 0 is the 4K byte Read Only Memory which is local to each Processor Node. It contains the bootstrap code and certain fundamental parts of the configuration and debugging system. The contents of the EPROM on all Processor Nodes are expected to be identical.

- Subspace 1 will contain the control registers on the I/O cards. This portion of the machine has not yet been fully specified.

- Subspace 2 contains the memory which is local to this Processor Node and which should be addressed directly.

- Subspace 3 contains remote memory. Local memory (subspace 2) is also contained in Subspace 3 but can be more efficiently accessed through Subspace 2. This distinction is addressed in more detail below.

- Subspace 4 contains the Memory Management Unit's control registers. These registers will be described later in this report.

- Subspaces 5, 6, and 7 are set aside for various control registers in the Processor Node itself.

One of the more important spaces is Subspace 2, where the physical memory for this Processor Node is located. As with all subspaces, it is limited in length to 1M bytes. This bounds the amount of memory that may be placed on a Processor Node. In addition, only 4 memory boards can be attached to a Processor Node. Since only 16K-bit memory IC's are presently available, the maximum amount of memory on one memory board is 128K bytes and as a result, the maximum amount of memory on a Processor Node is 512K bytes. The maximum amount of memory in a Butterfly Multiprocessor with 256 Processor Nodes is thus 128M bytes.

The 32-bit physical address uniquely represents each addressable memory and register in the machine. Its Processor Node, subspace, and subspace offset are fully specified. As we will see later, the hardware uses Subspace 3 in a special way which places limits on how a physical address is interpreted. Subspace 3 is not a subspace in the usual sense. Rather, it is a way of indicating to the hardware of a Processor Node that this reference is not to be interpreted as a local reference.

While the physical address space reflects the structure and needs of the hardware, the virtual address space reflects the structure and needs of the software. The software of the Butterfly Multiprocessor is divided into processes. Each process executes in its own virtual address space, although the virtual address spaces of several processes may reference the same physical memory, if desired. Thus, while there can only be one physical address space in a machine, there may be many virtual address spaces in each Processor Node.

The size of the virtual address space of the Butterfly Multiprocessor has been designed to match the processor being used. The MC68000 is, in a large sense, a 32-bit machine. As such, address registers within the machine have a capability for 32-bit operations. However, only the lower 24 bits of the address are actually supported in the current implementation of the MC68000. As a result, for our purposes, a virtual address is a 24-bit number.

In the Butterfly Multiprocessor we will treat the 24-bit virtual address as an 8-bit segment number followed by a 16-bit

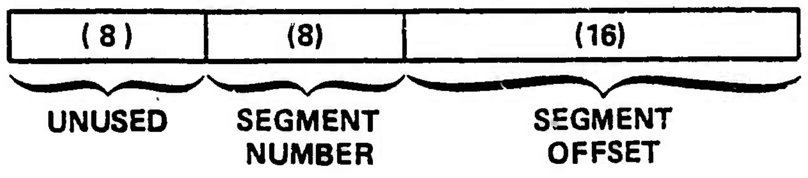

Figure 2.1-2 Virtual Address Format

offset within the segment, as shown in Figure 2.1-2. This divides the address space of the machine into 256 segments. Each segment can be from 256 to 64K bytes long. Each segment has individual protection and relocation attributes, and each may represent local memory (on the same Processor Node), remote memory, local I/O device registers, or local control registers.

The structure of the Processor Node is shown in Figure 2.1-3. This figure illustrates the distinction between the virtual and the physical address spaces. The processor (and as a result, the programmer) lives in a virtual address space. Every memory reference undergoes a translation into a physical address by the Memory Management Unit (MMU) before the access is performed. The Processor Node Controller, the switch, the I/O devices, and the memory live in a physical address space. The MMU links these two spaces. This separation of the machine into portions which operate in a virtual address space and those which operate in a physical address space seems straightforward, but brings forth many subtle issues.

There have been many views of what transformation is appropriate between the address generated and used by the processor and the address used by the memory hardware of a machine. The simplest is the identity transformation in which the virtual and physical address spaces are identical and no transformation is performed. This requires less hardware and imposes less delay in the path of a memory access than the current virtual/physical split does.

The identity transformation is also more efficient since there is less overhead associated with manipulation of the address space and context swapping overhead can be reduced.

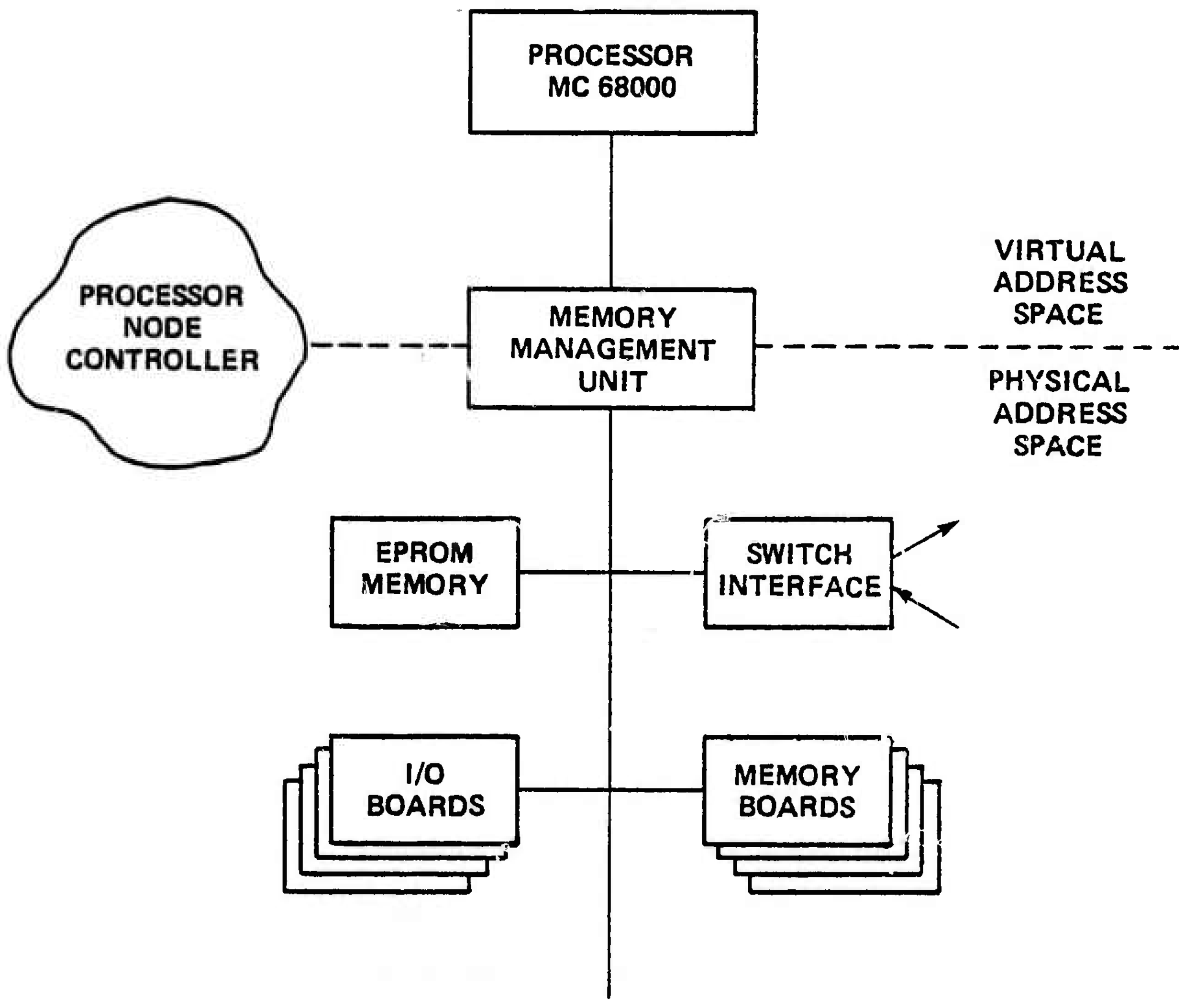

Figure 2.1-3 Processor Node Diagram

However, the most important issue involving the virtual memory scheme concerns the architecture of the machine.

A choice must be made between the use of virtual or physical addresses in the implementation of the I/O system because a DMA transfer must be programmed using either virtual or physical addresses. The transfer will be initiated by a user process running in a virtual environment, but when the DMA accesses occur, it may be hard to identify and locate the appropriate environment. Alternatively, if the DMA operates in the physical address space, it is necessary to convert the virtual addresses of the DMA pointers into physical addresses and to ensure that every DMA access is legal. Such accesses may not be legal even if the control blocks are valid.

The split between virtual and physical address space is even more complicated in a tightly coupled multiprocessor since the mapping function is performed only once. A process on one Processor Node executes in a controlled virtual address space. When it accesses a memory location on another Processor Node, the second Processor Node is a physical access. As a result, the second Processor Node must trust the first Processor Node to generate a legal and valid memory address. As a result, the Processor Nodes are vulnerable to hardware or software errors on other Processor Nodes.

To fix this problem, we could add another map which would protect and perhaps relocate accesses that come in over the Butterfly Switch. However, it is very difficult to manage this extra layer of indirection.

The goal of a virtual memory system is to allow the software to be better protected and more easily written. The virtual and physical address spaces have different constraints. The physical address space must be large enough that the ultimate main memory, I/O, and special register spaces can be represented. This space must be organized so that any physical address can be reached quickly.

The virtual address space, on the other hand, must be large enough that the program may be written without becoming "cramped". There is no reason to suppose that the physical and virtual spaces should be the same size or should be organized in the same way.

Having decided to implement a virtual memory system, we face the options of segmentation and paging. Segmentation is the division of an address space into variable length blocks, usually by means of a virtual memory mapping. Paging, on the other hand, divides the space into fixed length pieces. The implications of the two schemes are quite different. Segments are intended as an aid to the sophisticated programmer in organizing the protection, sharing, and location of his process. Pages are used to break up the full address space into more manageable fixed size pieces for the operating system. Often demand paging is used to "cache" the contents of a virtual address space through swapping on a secondary storage medium.

Segmentation can be implemented on top of paging, as in the Multics system. However, such a scheme appears too complex and requires too much additional mechanism. In addition, since the MC68000 cannot now support demand paging, we have determined that a simple segmentation scheme is most appropriate for the Butterfly Multiprocessor.

2.2 Address Transformation

The purpose of memory management is to separate the virtual address space as seen by the processing elements from the physical address space. The memory manager serves as an interface between these two spaces, translating virtual addresses into physical addresses. The time of the mapping is also a convenient one to perform some ancillary functions such as protection.

The memory management system of the Butterfly Multiprocessor contains 512 Segment Attribute (SA) registers on each Processor Node. Each SA register defines the address translation and protection characteristics for one segment. A set of registers are grouped together to form an address space for a process. There is also a single Address Space Attribute (ASA) register which, when loaded with the address and extent of a group of SA registers, defines the currently active address space.

This provides for holding many address spaces in the SA register table at the same time. The mapping registers of the MMU are built from 1K-bit by 4-bit static RAMS. Since the largest address space is 256 segments, and many processes will be less than 16 segments long, there is a question of what to do with the unused mapping registers. Since the stages of address translation and access control need not happen at the same time, the hardware time-multiplexes the mapping registers. This reduces the number of mapping registers to 512. This is still many more than the number of segments used by a process.

A virtual address space can consist of 4, 8, 16, 32, 64, 128, or 256 segments. Since there are 512 SA registers in the hardware of a Processor Node, enough Segment Attribute registers are available to hold from 2 to 128 address spaces simultaneously. In order to change address spaces, it is only necessary to change the ASA register. We hope that this will result in a very fast process switching time.

The virtual address translation function has several phases:

1. Using the ASA register and the segment number from the virtual address, select the correct SA register. If the segment number is invalid, give an error.

2. Using the SA register selected above and the segment offset from the virtual address, generate the physical address. If the segment offset is too large, give an error.

3. Using the SA register and the access mode of this memory reference (e.g., read vs write, instruction vs data) give an error if the access is illegal.

4. Process the access on the basis of its subspace field.

While this appears to be very simple and clean, the actual implementation in the hardware imposes several constraints. This implementation is shown in more detail in Figure 2.2-1. The address transformation function is defined as follows:

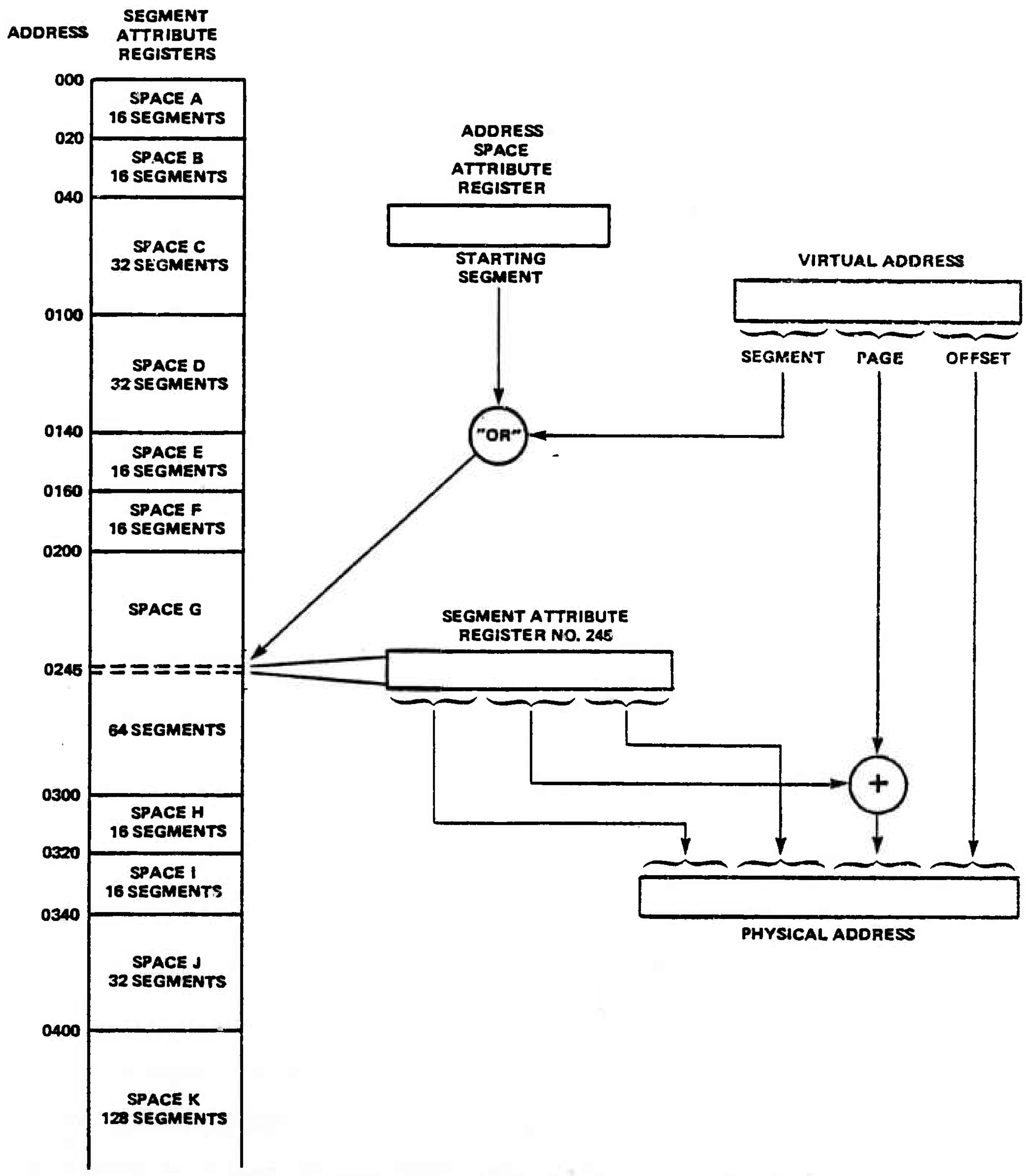

Figure 2.2-1 Virtual to Physical Address Translation

The ASA register is a 12-bit register. The three most significant bits contain a code which specifies how many segments are in this process's address space. If a reference is made to a segment whose number exceeds this limit, an error will be generated. Since not all address space sizes can be specified by an ASA code, it may be necessary to specify a larger than required segment count. Unused segments may then be marked as invalid in the SA registers.

The least significant 9 bits of the ASA register are logically "ORed" with the 8-bit segment number (padded with a zero on the left) to select the correct SA register. During this logical "OR", the bottom two bits of the ASA register are ignored and presumed to be zero.

The logical "OR" function is used instead of addition because it is more quickly and easily calculated by the hardware. However, it constrains the allocation of SA registers. This issue is discussed further in Section 2.3.2.

Once the correct SA register has been selected, it is combined with the segment offset from the virtual address to form the correct physical address. As the figure illustrates, the least significant 8 bits of the physical address come directly from the least significant 8 bits of the virtual address, while the most significant 16 bits come from the SA register directly.

The bits in between are the sum of fields from the SA register and the segment offset. This is detailed in the table below:

Physical Address Bits	Derived From
7-0	Bits 7-0 of the virtual address
15-8	Sum of bits 15-8 of the virtual address and bits 15-8 of the SA Register
23-16	Bits 7-0 of the SA Register
31-24	Bits 31-24 of the SA Register

Since the segment offset in a virtual address is 16 bits long, the largest segment is 64K bytes. However, segments need not be this large. A segment length field in the SA register defines the actual length. Each segment is defined to start at an offset of zero and to increase to this limit. The table below gives the segment length code and the corresponding limit on the number of bytes in the segment. The number is given in both decimal and octal for convenience. The largest valid segment offset is one less than this limit.

Segment Length Code	Segment Offset Limit (Decimal)	(Octal)
0	256	0400
1	65536	0200000
2	512	01000
3	768	01400
4	1024	02000
5	1536	03000
6	2048	04000
7	3072	06000
8	4096	010000
9	6144	014000
10	8192	020000
11	12288	030000
12	16384	040000
13	24576	060000
14	32768	0100000
15	49152	0140000

For each virtual-to-physical address translation, the MMU also checks that the access is being performed in a legal mode. This protection is on a segment-by-segment basis. All of the locations in a segment are protected identically. The protection mechanism checks the access according to whether the processor is in Supervisor or User mode and whether the access is a data read or write or an instruction fetch. The code which specifies which modes of access are legal is in the SA register for the segment. The table below gives the protection codes and the corresponding set of legal access modes:

Protection Class	Allowed Access Characteristics
0	Illegal to access
1	Read only data or instruction fetch
2	All accesses are legal
3	Read or write data
4	Read only data
5	Supervisor read or write data
6	Supervisor read only data
7	Supervisor data or User data read only

If the access is legal, the access sequences to completion. If an inconsistency is detected, a bus error will be signaled to the MC68000 and the access will be aborted.

Once the correct physical address has been generated and its validity checked, it is necessary to reference the correct physical memory location. While the physical address can uniquely specify the physical location, the hardware takes a short cut.

The hardware interprets the physical address first on the basis of the subspace field rather than the Processor Node number. If the subspace is anything but subspace 3 (implying a remote memory reference), the access proceeds without regard for the Processor Node number. If the access is to subspace 3, a switch message is created even if the Processor Node number refers to the local Processor Node. This is important in achieving the highest possible memory speed and simplifies the hardware. For consistency, we expect the software to maintain the correct values in the Processor Node field even where the hardware would ignore it. Similarly, the software should normally detect when a memory segment is local to this Processor Node, and should declare it to be in Subspace 2.

As a result of this implementation, the only subspace which may be accessed across the switch is the memory on a Processor Node. It is not possible to generate an access to a remote Processor Node's I/O device registers, its Processor Node Controller registers, or its EPROM.

It is possible to send messages out through the switch and back to oneself simply by referencing subspace 3 with the correct Processor Node number. This will be useful in testing this Processor Node's switch interface and a portion of the switch.

2.3 Design Considerations

The Memory Management hardware implements a virtual memory system as described above. The design of any virtual memory system raises complex issues. This section discusses some of the issues which went into this design and elaborates some areas which require special care in the software which interfaces to the memory system.

2.3.1 Supervisor and User Modes

The Memory Management Unit implements the virtual address space of a process. This leads to the question: What is the structure of a process's address space? The final use of the virtual address space is certainly less well defined than much of the hardware discussed previously, but nonetheless, it is important to consider an example in hopes of understanding the overall structure. The following example represents how the address space of one process might be laid out.

Segment	Contents
0	Interrupt Vectors
1	Supervisor Code
2	Supervisor Data
3	Supervisor Stack
4	Memory Management Registers
5	User Code
6	User Local Data
7	User Stack
8	User Auxiliary Segment

Segments 0 through 4 are for the supervisor. This points up a very important difference between this machine and other supervisor/user designs. The supervisor code does not run in a different address space from the user code, but rather runs in a subset of the same address space, but with protection modes so that this subset is not accessible in user mode.

This structure has several important advantages. First, it permits a very natural form of argument passing between the user and the supervisor since the user mode arguments are also in the supervisor's address space. This is particularly true for arguments which are pointers to structures in the user's environment.

Second, it makes it very natural to think of the supervisor as running on behalf of a particular user. There are separate stacks and separate data regions (if desired) for each process. Where it is necessary for the supervisor of one process to synchronize with another, this structure is very natural.

The structure also has some disadvantages. It requires more mapping registers, since the supervisor is present in every address space. Presumably, with more hardware and a slightly slower memory management system, some of this overhead could be saved. It also requires that this portion of every process be the same.

2.3.2 Multiple Address Spaces

The use of processes in a real-time environment requires that the time spent scheduling processes and changing from one process to another be short. In order to minimize the process switching time, we have designed the hardware so that the maps for many processes can be in the mapping registers at the same time. One need only change the pointer into the mapping registers (the ASA Register) to change the current address space. For example, it is possible to have 32 processes which are 16 segments long in the mapping registers at the same time.

An obvious question arises: What if there are more processes than fit in the maps? The simple answer is to swap the maps into and out of the SARs. However, since the number of processes in existence at any one time is probably limited by the amount of memory on a Processor Node, we can defer the design of the map swapping algorithms until we have a similar mechanism for swapping memory itself.

Unfortunately, the ASA Register cannot point to any arbitrary map location because the mapping register which will be used is not the sum of the ASA and the segment being addressed, but rather the "OR" of the two. This means that it is equivalent to addition only if there are no carries.

In practice, this requires that the rightmost N bits of the ASA Register be zero for a space of 2**N segments. This means that address spaces must start on power-of-2 boundaries. One must of course manage the use of these memory maps, placing new processes in the maps and removing processes when they are destroyed. One of the simpler management schemes is the "buddy system". Fortunately, this allocation algorithm is itself based on power-of-2 boundaries and as a result matches the hardware restriction very well.

A similar restriction occurs in that the adder in the translation does not carry past 16 bits in the physical memory space. This means that there are boundaries in physical memory every 2**16 (or 64K) bytes which segments may not cross. This boundary need not be visible to the application software, and is easily hidden by the physical memory allocation software: since the physical memory allocator will not allocate a block of memory which crosses this boundary, segments cannot be constructed which cross it.

2.3.3 Protection

The MMU not only defines a virtual address space but also controls accesses to it. The primitive unit of protection is the segment. Each segment can be protected so that certain subsets of accesses are permitted. The dimensions of this access are whether the processor is in User or Supervisor mode at the time and whether the access is an instruction fetch or a data read or write. There are 6 possible access modes, and therefore 6 bits are necessary to specify any set of valid access modes. Only a subset of the possible access modes have been implemented in the 3 bits which code the permitted accesses. We may or may not have selected the correct options, but we have placed the protection checking in programmable hardware which is very easily changed.

The MMU also permits control over the length of a segment. In particular, the mapping register for each segment specifies how many pages (of 256 bytes) are contained in a segment. The available options span the range from 1 to 256 pages in nonuniform increments as shown in Section 2.2. The difference in size between two successive limits is approximately half of the smaller limit. This scheme is obviously a bit coarse since the smallest object that can be controlled is 256 bytes long and large objects will rarely have bounds which accurately match their length.

In spite of these limitations, the Memory Management philosophy of the Butterfly is quite sophisticated. The impact of this on the complexity of the hardware is only now becoming clear, and its effect on the execution speed of the machine is yet to be determined. Finally, such a sophisticated mechanism is included in hopes that the difficult debugging problems which occur so frequently in multiprocessor systems would be detected by this hardware. We cannot yet tell whether this advantage outweighs the additional complexity of the software which manages protection.

2.3.4 Initialization

At power-up, the contents of the various memory management registers are undefined. Since the processor cannot execute without these registers, immediately following a power-up sequence or a restart message (sent from another Processor Node), the Processor Node hardware will initialize the ASA register to indicate an 8 segment address space with the following segment attributes:

Segment Number	Access Code	Node Number	Sub-Space	Segment Offset	Segment Length
0	2	N	0	0	256
1	2	N	1	0	256
2	2	N	2	0	256
3	2	N	3	0	256
4	2	N	4	0	256
5	2	N	5	0	256
6	2	N	6	0	256
7	2	N	7	0	256

In this table, N is the local Processor Node number. Thus all subspaces permit all types of accesses. After the Processor Node initializes various other Processor Node registers it resets the MC68000 which then responds by reading th first four words in virtual memory space (mapped into the first four words in EPROM) and loading them into the supervisor's stack pointer and program counter. The stack pointer will be initialized to a location in segment four and the program counter will be initialized to a location in segment zero. The MC68000 will initialize the status register to an interrupt level of seven.

It is hoped that this initialization will be realized by a very simple loop in the Processor Node Controller. If not, another initialization scheme may be devised.

DISTRIBUTION OF THIS REPORT

Defense Advanced Research Projects Agency

Dr. Robert E. Kahn (2)

Defense Supply Service -- Washington

Jane D. Hensley (1)

Defense Documentation Center (12)

Bolt Beranek and Newman Inc.

Library

Library, Canoga Park Office

R. Bressler

R. Brooks

P. Carvey

P. Castleman

G. Falk

F. Heart

M. Hoffman

M. Kraley

W. Mann

J. Pershing

R. Rettberg

E. Starr

E. Wolf